Knowledge Matters: Importance of Prior Information for Optimization

Çağlar Gülçehre and Yoshua Bengio
Département d'informatique et de recherche opérationnelle
Université de Montréal, Montréal, QC, Canada

Abstract

We explore the effect of introducing prior information into the intermediate level of neural networks for a learning task on which all the state-of-the-art machine learning algorithms tested failed to learn. We motivate our work from the hypothesis that humans learn such intermediate concepts from other individuals via a form of supervision or guidance using a curriculum. The experiments we have conducted provide positive evidence in favor of this hypothesis. In our experiments, a two-tiered MLP architecture is trained on a dataset with 64x64 binary inputs images, each image with three sprites. The final task is to decide whether all the sprites are the same or one of them is different. Sprites are pentomino tetris shapes and they are placed in an image with different locations using scaling and rotation transformations. The first part of the two-tiered MLP is pre-trained with intermediate-level targets being the presence of sprites at each location, while the second part takes the output of the first part as input and predicts the final task's target binary event. The two-tiered MLP architecture, with a few tens of thousand examples, was able to learn the task perfectly, whereas all other algorithms (include unsupervised pre-training, but also traditional algorithms like SVMs, decision trees and boosting) all perform no better than chance. We hypothesize that the learning difficulty involved when not pre-training with intermediate targets is due to the *composition* of two highly non-linear tasks. Our findings are also consistent with hypotheses on cultural learning inspired by the observations of local minima problems with deep learning, presumably because of effective local minima.

1 Introduction

There is a recent emerging interest in different fields of science for *cultural learning* (Henrich and McElreath, 2003) and how groups of individuals exchanging information can learn in ways superior to individual learning. This is also witnessed by the emergence of new research fields such as "Social Neuroscience". Learning from other agents in an environment by the means of cultural transmission of knowledge with a peer-to-peer communication is an efficient and natural way of acquiring or propagating common knowledge. The most popular belief on how the information is transmitted between individuals is that bits of information are transmitted by small units, called memes, which share some characteristics of genes, such as self-replication, mutation and response to selective pressures (Dawkins, 1976).

This paper is based on the hypothesis (which is further elaborated in Bengio (2013)) that human culture and the evolution of ideas have been crucial to counter a local minima issue: this difficulty would otherwise make it intractable for human brains to capture high level knowledge of the world. Here we use machine learning experiments to investigate some elements of this hypothesis by seeking answers for the following questions: are there machine learning tasks which are intrinsically hard for a lone learning agent but that may become very easy when intermediate concepts are provided by

another agent as additional intermediate learning cues, in the spirit of Curriculum Learning (Bengio *et al.*, 2009a)? What makes such learning tasks more difficult? Can we verify that we are dealing with a local minima issue of deep networks, i.e., that using the same architecture but only changing initial conditions can change the outcome from complete success to complete failure? These are the questions discussed (if not completely addressed) here, which relate to the following broader question: how can humans (and potentially one day, machines) learn complex concepts?

In this paper, we present results on an artificial learning task involving binary 64×64 images. Each image in the dataset contains 3 pentomino tetris sprites (simple shapes). The task is to figure out if all the sprites in the image are the same or if there are different sprite shapes in the image. We have tested several state-of-the-art machine learning algorithms and none of them could perform better than a random predictor on the test set. Nevertheless by providing hints about the intermediate concepts (the presence and location of particular sprite classes), the problem can easily be solved where the same-architecture neural network without the intermediate concepts guidance fails. Surprisingly, our attempts at solving this problem with unsupervised pre-training algorithms failed solve this problem. For showing the impact of intermediate level guidance, we experimeted with a two-tiered neural network, with supervised pre-training of the first part to recognize the category of sprites independently of their orientation and scale, at different locations, while the second part learns from the output of the first part and predicts the binary task of interest.

Of course, the objective is not to propose a novel learning algorithm or architecture, but rather to refine our understanding of the learning difficulties involved with composed tasks (here a logical formula composed with the detection of object classes), in particular the training difficulties involved for deep neural networks. The results also bring empirical evidence in favor of some of the hypotheses from Bengio (2013), discussed below, as well as introducing a particular form of curriculum learning (Bengio *et al.*, 2009a).

1.1 Curriculum Learning and Cultural Evolution Against Local Minima

What Bengio (2013) calls an **effective local minimum** is a point where iterative training stalls, either because of an actual local minimum or because the optimization algorithm is unable (in reasonable time) to find a descent path (e.g., because of serious ill-conditioning).

The idea that learning can be enhanced by guiding the learner through intermediate easier tasks is old, starting with animal training by *shaping* (Skinner, 1958; Peterson, 2004; Krueger and Dayan, 2009). Bengio *et al.* (2009a) introduce a computational hypothesis related to a presumed local minima issue when directly learning the target task: the good solutions correspond to hard-to-find-by-chance effective local minima, and intermediate tasks prepare the learner's internal configuration (parameters) in a way similar to continuation methods in global optimization (which go through a sequence of intermediate optimization problems, starting with a convex one where local minima are no issue, and gradually morphing into the target task of interest).

In a related vein, Bengio (2013) makes the following inferences based on experimental observations of deep learning and neural network learning:

Point 1: Training deep architectures is easier when some hints are given about the function that the intermediate levels should compute (Hinton *et al.*, 2006; Weston *et al.*, 2008; Salakhutdinov and Hinton, 2009; Bengio, 2009). *The experiments performed here expand in particular on this point.*

Point 2: It is much easier to train a neural network with supervision (where we provide it examples of when a concept is present and when it is not present in a variety of examples) than to expect unsupervised learning to discover the concept (which may also happen but usually leads to poorer renditions of the concept).

Point 3: Directly training all the layers of a deep network together not only makes it difficult to exploit all the extra modeling power of a deeper architecture but in many cases it actually yields worse results as the number of *required layers* is increased (Larochelle *et al.*, 2009; Erhan *et al.*, 2010). *The experiments performed here also reinforce that observation.*

Point 4: Erhan *et al.* (2010) observed that no two training trajectories end up in the same local minimum, out of hundreds of runs. This suggests that the number of functional local minima

(i.e. corresponding to different functions, each of which possibly corresponding to many instantiations in parameter space) must be very large.

Point 5: Unsupervised pre-training, which changes the initial conditions of the descent procedure, sometimes allows to reach substantially better local minima (in terms of generalization error!), and these better local minima do not appear to be reachable by chance alone (Erhan *et al.*, 2010). *The experiments performed here provide another piece of evidence in favor of explanatory hypotheses based on a training difficulty due to local minima.*[1]

Based on the above points, Bengio (2013) then proposed the following hypotheses regarding learning of high-level abstractions.

- **Optimization Hypothesis:** When it learns, a biological agent performs an approximate optimization with respect to some implicit objective function.
- **Deep Abstractions Hypothesis:** Higher level abstractions represented in brains require deeper computations (involving the composition of more non-linearities).
- **Local Descent Hypothesis:** The brain of a biological agent relies on approximate local descent and gradually improves itself while learning.
- **Effective Local Minima Hypothesis:** The learning process of a single human learner (not helped by others) is limited by effective local minima.
- **Deeper Harder Hypothesis:** Effective local minima are more likely to hamper learning as the required depth of the architecture increases.
- **Abstractions Harder Hypothesis:** High-level abstractions are unlikely to be discovered by a single human learner by chance, because these abstractions are represented by a deep subnetwork of the brain, which learns by local descent.
- **Guided Learning Hypothesis:** A human brain can learn high level abstractions if guided by the signals produced by other agents that acts as hints or indirect supervision for these high-level abstractions.
- **Memes Divide-and-Conquer Hypothesis:** Linguistic exchange, individual learning and the recombination of memes constitute an efficient evolutionary recombination operator in the meme-space. This helps human learners to *collectively* build better internal representations of their environment, including fairly high-level abstractions.

This paper is focused on "*Point 1*" and testing the "*Guided Learning Hypothesis*", using machine learning algorithms to provide experimental evidence. The experiments performed also provide evidence in favor of the "*Deeper Harder Hypothesis*" and associated "*Abstractions Harder Hypothesis*". Machine Learning is still far beyond the current capabilities of humans, and it is important to tackle the remaining obstacles to approach AI. For this purpose, we are particularly interested in tasks that humans learn effortlessly from very few examples, while machine learning algorithms fail miserably.

2 Culture and Optimization Difficulty

As hypothesized in the "*Local Descent Hypothesis*", human brains would rely on a local approximate descent, just like a Multi-Layer Perceptron trained by a gradient-based iterative optimization. The main argument in favor of this hypothesis relies on the biologically-grounded assumption that although firing patterns in the brain change rapidly, synaptic strengths underlying these neural activities change only gradually, making sure that behaviors are generally consistent across time. If a learning algorithm is based on a form of local (e.g. gradient-based) descent, it can be sensitive to local minima (Bengio, 2013). Note that throughout this paper, when talking about "local minima", we refer to the local minima of the generalization error. We are mostly interested in the online setting where the online gradient (associated with the next example) is an unbiased estimator of the gradient of generalization error.

[1] Recent work showed that rather deep feedforward networks can be very successfully trained when large quantities of labeled data are available (Ciresan *et al.*, 2010; Glorot *et al.*, 2011a; Krizhevsky *et al.*, 2012). Nonetheless, the experiments reported here suggest that it all depends on the task being considered, since even with very large quantities of labeled examples, the deep networks trained here were unsuccessful.

When one trains a neural network, at some point in the training phase the evaluation of error seems to saturate, even if new examples are introduced. In particular Erhan *et al.* (2010) find that early examples have a much larger weight in the final solution. It looks like the learner is stuck in or near a local minimum. But since it is difficult to verify if this is near a true local minimum or simply an effect of strong ill-conditioning, we call such a "stuck" configuration an *effective local minimum*, whose definition depends not just on the optimization objective but also on the limitations of the optimization algorithm.

Erhan *et al.* (2010) highlighted both the issue of effective local minima and a regularization effect when initializing a deep network with unsupervised pre-training. Interestingly, as the network gets deeper the effect of local minima seems to be get more pronounced. That might be because of the number of local minima increases, or maybe the good ones are harder to reach.

As a result of Point 4 we hypothesize that it is very difficult for an individual's brain to discover some higher level abstractions by chance only. As mentioned in the "*Guided Learning Hypothesis*" humans get hints from other humans and learn high-level concepts by the guidance of other humans[2]. Curriculum learning (Bengio *et al.*, 2009b) and incremental learning (Solomonoff, 1989), are examples of this. This is done by properly choosing the sequence of examples seen by the learner, where simpler examples are introduced first and more complex examples shown when the learner is ready for them. The hypothesis about why curriculum works states that curriculum learning acts as a continuation method that allows one to discover a good minimum, by first finding a good minimum of a smoother error function. Recent experiments on human subjects also shows that humans *teach* using a curriculum strategy (Khan *et al.*, 2011).

Some parts of the human brain are known to have a hierarchical organization (i.e. visual cortex) consistent with the deep architecture studied in machine learning papers. As we go from the sensory level to higher levels of the visual cortex, we find higher level areas corresponding to more abstract concepts. This is consistent with the *Deep Abstractions Hypothesis*.

Training neural networks and machine learning algorithms by decomposing the learning task into sub-tasks and exploiting prior information about the task is well-established and in fact constitutes the main approach to solving industrial problems with machine learning. The contribution of this paper is rather on rendering the local minima issue explicit and providing evidence on the type of problems for which this difficulty arises. This prior information and hints given to the learner can be viewed as inductive bias for a particular task, an important ingredient to obtain a good generalization error (Mitchell, 1980). An interesting earlier finding in that line of research was done with Explanation Based Neural Networks (EBNN) in which a neural network transfers knowledge across multiple learning tasks. An EBNN uses previously learned domain knowledge as an initialization or search bias (i.e. to constrain the learner in the parameter space) (O'Sullivan, 1996; Mitchell and Thrun, 1993).

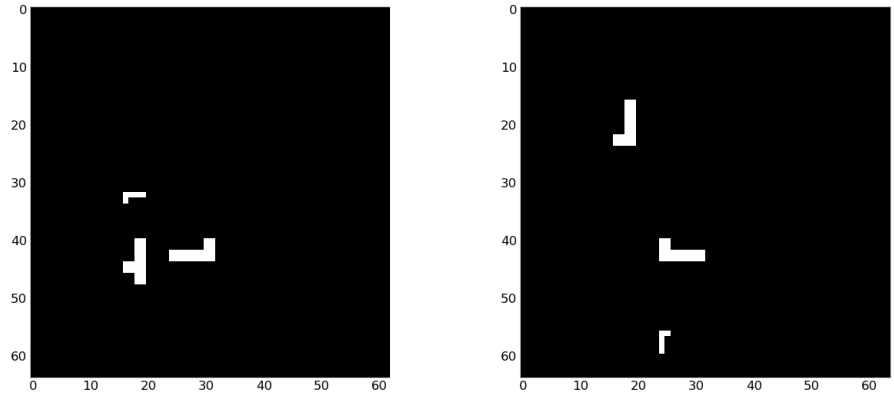
Another related work in machine learning is mostly focused on reinforcement learning algorithms, based on incorporating prior knowledge in terms of logical rules to the learning algorithm as a prior knowledge to speed up and bias learning (Kunapuli *et al.*, 2010; Towell and Shavlik, 1994).

3 Experimental Setup

Some tasks, which seem reasonably easy for humans to learn[3], are nonetheless appearing almost impossible to learn for current state-of-art machine learning algorithms. Here we study more closely such a task, which becomes learnable if one provides to the learner hints about appropriate intermediate concepts. Interestingly, the task we used in our experiments is not only hard for deep neural networks but also for non-parametric machine learning algorithms such as SVMs, boosting and decision trees.

[2]But some high-level concepts may also be hardwired in the brain, as assumed in the universal grammar hypothesis (Montague, 1970), or in nature vs nurture discussions in cognitive science.

[3]keeping in mind that humans can exploit prior knowledge, either from previous learning or innate knowledge).



(a) sprites, not all same type

(b) sprites, all of same type

Figure 1: Left (a): An example image from the dataset which has *a different sprite type* in it. Right (b): An example image from the dataset that has only one type of pentomino object in it, but with different orientations and scales.

3.1 Pentomino Dataset

In order to test our hypothesis, we designed an artificial dataset for object recognition using 64×64 binary images[4]. If the task is two tiered (i.e., with guidance provided), the task in the first part is to recognize and locate each pentomino object class[5] in the image. The second part and final task is to figure out if all the pentominos in the image are of the same class or not. Hence after a neural network learned the categories of each object in the dataset, the task becomes an XOR-like operation between the object categories detected in the image. The types of pentomino objects that we have used for generating the dataset are as follows: Pentomino sprites N, P, F, Y, J, and Q, along with the Pentomino N2 sprite (mirror of "Pentomino N" sprite), the Pentomino F2 sprite (mirror of "Pentomino F" sprite), and the Pentomino Y2 sprite (mirror of "Pentomino Y" sprite).

As shown in Figures 1(a) and 1(b), the synthesized images are fairly simple and do not have any texture. Foreground pixels are "1" and background pixels are "0". Images of the training and test sets are generated iid. For notational convenience, assume that the domain of raw input images is X, the set of sprites is S, the set of intermediate object categories is Y for each possible location in the image and the set of final task outcomes is Z. We perform two different types of rigid body transformation: sprite rotation $rot(X, \gamma)$ where $\Gamma = \{\gamma\colon (\gamma = 90 \times \phi) \wedge [(\phi \in \mathbb{N}), (0 \leq \phi \leq 3)]\}$ and scaling $scale(X, \alpha)$ where $\alpha \in \{1, 2\}$ is the scaling factor. The data generating procedure is summarized below.

Sprite transformations: Before placing the sprites in an empty image, for each image $x \in X$, we randomly decide on the $z \in Z$ to have (or not) a different sprite in the image. Conditioned on the constraint given by z, we randomly select three sprites s_{ij} from S without replacement. Using a uniform probability distribution over all possible scales, we choose a scale and accordingly scale each sprite image. Then we randomly rotate each sprite by a multiple of 90 degrees.

Sprite placement: Upon completion of sprite transformations, we generate a 64×64 uniform grid divided into 8×8 blocks, each block being of size 8×8 pixels, and randomly select three

[4]The source code for the script that generates the artificial pentomino datasets (Arcade-Universe) is available at: `https://github.com/caglar/Arcade-Universe`. This implementation is based on Olivier Breuleux's bugland dataset generator.

[5] A human learner does not seem to be taught the shape categories of each pentomino sprite in order to solve the task. On the other hands, humans have lots of previously learned knowledge about the notion of shape and how central it is in defining categories.

different blocks from the 64=8×8 on the grid and place the transformed objects into *different* blocks (so they cannot overlap, by construction).

Each sprite is centered in the block in which it is located. Thus there is no object translation inside the blocks. The only translation invariance is due to the location of the block inside the image.

A pentomino sprite is guaranteed to not to overflow the block in which it is located, and there are no collisions or overlaps between sprites, making the task simpler. The largest possible pentomino sprite can be fit into an 8×4 mask.

3.2 Learning Algorithms Evaluated

We have first cross-validated our models by using 5-fold cross-validation. With 40,000 examples, this gives 32,000 examples for training and 8,000 examples for testing. For neural network algorithms, we have used stochastic gradient descent (SGD) for training. The following standard learning algorithms were evaluated: decision trees, SVMs with Gaussian kernel, ordinary fully-connected Multi-Layer Perceptrons, Random Forests, k-Neareset Neighbors, Convolutional Neural Networks, and Stacked Denoising Autoencoders with supervised fine-tuning. More details of the configurations and hyper-parameters for each of them are given in Appendix, section 6.1.

3.2.1 Intermediate Knowledge Guided Neural Network (IKGNN)

The IKGNN is a two-part deep neural network, in which the first part's training objective is the detection and classification of the pentomino sprite classes in an 8×8 patch. The Part 1 Neural Net (P1NN) is applied to each of the 8×8=64 non-overlapping patches of the 64×64 input image, in order to produce the input for the Part 2 Neural Net (P2NN). P1NN is trained with the intermediate target Y. Y specifies the type of (if any) pentomino sprite present for each of the 64 patches (8×8 non-overlapping blocks) of the image. Because a possible answer at a given location can be "none of the object types" i.e., an empty patch, y_p (for patch p) can take 11 values, 1 for rejection and the rest is for the 10 different pentomino classes:

$$y_p = \begin{cases} 0 & \text{if patch } p \text{ is empty} \\ s \in S & \text{if the patch } p \text{ contains a pentomino sprite .} \end{cases}$$

A similar task has been studied by Fleuret *et al.* (2011) (SI appendix Problem 17) and they compared performance of humans and computers.

The IKGNN architecture is a two tiered network that takes advantage during training of prior information about intermediate-level relevant factors. Because the sum of the training losses decomposes into the loss on each patch, the P1NN can be pre-trained patch-wise. Each patch-specific component of the P1NN is a fully connected MLP with 8×8 inputs and 11 outputs with a softmax output layer. Formally let's assume that the function $f_\theta(\mathbf{x})$ computes the softmax output of P1NN , $z(a) = \frac{a-\mu_a}{\sigma_a}$ is the standardization function of variable a. The input x_{P2NN} of the P2NN will be:

$$x_{P2NN} = z(f_\theta(\mathbf{p_0}) * ... * f_\theta(\mathbf{p_i}) * ... * f_\theta(\mathbf{p_{63}})),$$

given that the $\mathbf{p_i}$ is the ith patch of the image and $*$ is the concatenation operation.

As seen on Figure 2 we trained the P1NN with respect to the intermediate target values (Y) and concatenate these outputs (for all the patches) into a one large vector (64×11). Then we standardize this output vector by subtracting its mean (over training examples) and dividing by its standard deviation (over training examples) for each element of the output vector. This normalized output vector of P1NN (of length 704) is then fed to the P2NN MLP, which has a single binomial output unit for the final task probability prediction (with a sigmoid unit) for the binary event Z, as seen on Figure 3.

IKGNN uses rectifier hidden units as activation function, $max(0,\ X)$, as in Jarrett *et al.* (2009); Nair and Hinton (2010); Glorot *et al.* (2011a); Krizhevsky *et al.* (2012). We found a significant boost by using rectification compared to hyperbolic tangent and sigmoid activation functions. The P1NN has a highly overcomplete architecture with 1024 hidden units per patch, and L1 and L2 weight decay regularization coefficients on the weights (not the biases) are respectively 1e-6 and 1e-5. The learning rate for the P1NN is 0.75. 2 training epochs were enough for the P1NN to learn the features

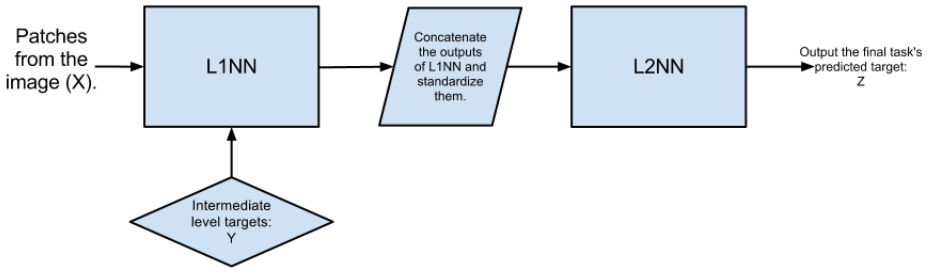


Figure 2: Information flow diagram for the IKGNN. P1NN is trained on the patches with respect to intermediate target labels and P2NN is trained on the concatenated and standardized output of P1NN with respect to the final task's target labels.

of the first layer. The P2NN has 2048 hidden units. L1 and L2 penalty coefficients for the P2NN are 1e-6 with a learning rate of 0.1. These were selected by trial and error based on validation set error. Both P1NN (patch-wise) and P2NN are fully-connected neural networks.

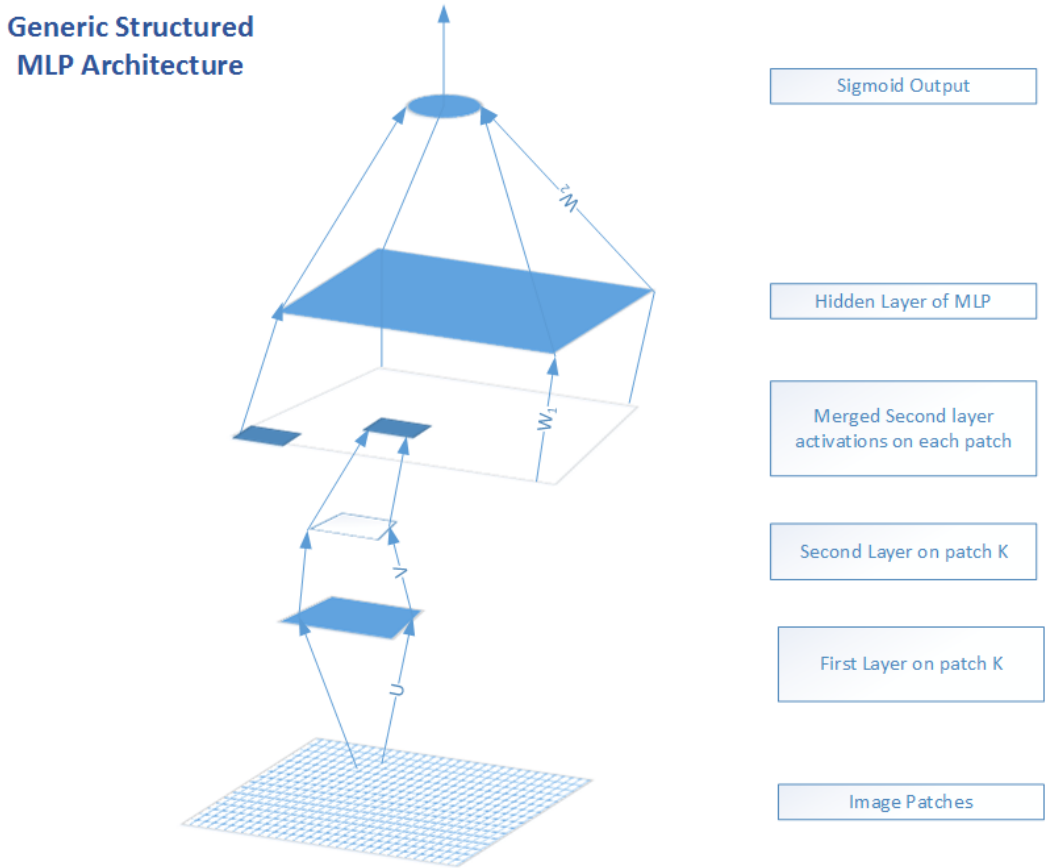


Figure 3: Structured MLP architecture, used for IKGNN (trained in two phases, first P1NN, bottom two layers, then P2NN, top two layers), and for the other structured MLP experiments. For IKGNN, P1NN is trained on each 8x8 patch extracted from the image and the softmax output probabilities of all 64 patches are concatenated into a 64x11 vector that forms the input of P2NN.

3.2.2 Deep and Structured Supervised MLP without Hints

We have used the same connectivity pattern (and deep architecture) that we used for the IKGNN but without using the intermediate targets (Y) and we directly predict the final outcome of the task (Z) by using the same number of hidden units, same connectivity and same activation function for the hidden units. We have evaluated 120 hyperparameter values by randomly selecting the number

SIZE OF LOCALLY CONNECTED LAYER	TRAINING ERROR	TEST ERROR
11	0.470	0.501
50	0.470	0.502
100	0.496	0.493

Table 1: The training and test error rates with different number of hidden units at the output of locally connected layer for structured MLP on the pentomino dataset.

of hidden units from $[64, 128, 256, 512, 1024, 2048]$, L1 penalty (on weights) coefficient from $[1e-6, 1e-5, 1e-4]$, L2 penalty (on weights) coefficient from $[1e-5, 1e-4, 1e-3, 1e-2]$ and randomly sampled 20 learning rates uniformly in the log-domain within the interval $[0.008, 0.8]$. We used two fully connected hidden layers with 1024 hidden units (same as P1NN) per patch and 2048 (same as P2NN) for the last hidden layer, with twenty training epochs. For this network we have obtained the best results with learning rate 0.05.[6]

For the structured MLP without hints (SMLP), we evaluated the effect of number of hidden units in the intermediate layer with number of hidden units in Table 1. In these experiments we have trained SMLP for 25 epochs on 100k samples. As a result of these experiments we could not see any significant effect of changing the number of hidden units in the locally connected layer on patches.

We also trained the Structured MLP with only 3 patches instead of the 64, by keeping only the patches containing an object. The model was trained using 100k examples, for 120 epochs. We have used a rectifier nonlinearity as the output of the locally connected layer and tried different numbers of hidden units. The lowest training error we have obtained was 37 percent but the best test error we got was still 49.2 percent. It was one way to try to make the task simpler, but did not impact the difficulty of training.

3.2.3 Deep and Structured MLP with Unsupervised Pre-Training

We have performed experiments similar as above, but by adding unsupervised pre-training, using denoising auto-encoder and/or contractive auto-encoders to pre-train each layer of P1NN. Supervised fine-tuning proceeds as in the deep and structured MLP without hints. We have also explored larger number of hidden units at the output of part 1, since previous work on unsupervised pre-training generally found that larger hidden layers were optimal when using unsupervised pre-training (because not all unsupervised features will be relevant to the task at hand). Instead of limiting to 11 units per patch, we experimented with networks with up to 20 hidden (i.e., code) units in the second-layer patch-wise auto-encoder.

We used Contractive Autoencoder(CAE) in the first layer with sigmoid nonlinearity and binary cross entropy cost function. In the second layer we have used Denoising Autoencoder with rectifier hidden units using L1 sparsity and weight decay on the weights of the autoencoder. In our experiments we used tied weights with autoencoders. We tried different combinations of CAE and DAE for unsupervised pretraining but none of the configurations we have tried manage to learn the Pentomino task.

3.3 Experiments with 1 of K representation

To explore the effect of changing the complexity of the model we have designed a set of experiments with symbolic representations of the information in each patch. In all cases an empty patch is represented with a 0 vector.

We have conducted 4 experiments by using following representations for each patch:

Experiment 1-Onehot representation without transformations: In this experiment we have done trials with a 10-input one-hot vector per patch. Each input corresponds to an object category.

[6]The source code of the structured MLP is available at the github repository: `https://github.com/caglar/structured_mlp`

Algorithm	20k dataset		40k dataset		80k dataset	
	Training Error	Test Error	Training Error	Test Error	Training Error	Test Error
SVM RBF	26.2	50.2	28.2	50.2	30.2	49.6
KNN	24.7	50.0	25.3	49.5	25.6	49.0
Decision Tree	5.8	48.6	6.3	49.4	6.9	49.9
Randomized Trees	3.2	49.8	3.4	50.5	3.5	49.1
MLP	26.5	49.3	33.2	49.9	27.2	50.1
CNN	50.6	49.8	49.4	49.8	50.2	49.8
2 layer sDA	49.4	50.3	50.2	50.3	49.7	50.3
Struct. supervised MLP	50.5	49.9	49.8	49.7	49.7	50.3
Struct. MLP+CAE Supervised Finetuning	50.54	49.68	49.75	49.68	50.25	49.68
Struct. MLP+CAE+DAE Supervised Finetuning	49.055	49.66	49.387	49.72	50.065	49.68
Struct. MLP+DAE+DAE Supervised Finetuning	50.54	49.68	49.745	49.68	50.249	49.68
IKGNN	**0.21**	**30.7**	**0**	**3.1**	**0**	**0.01**

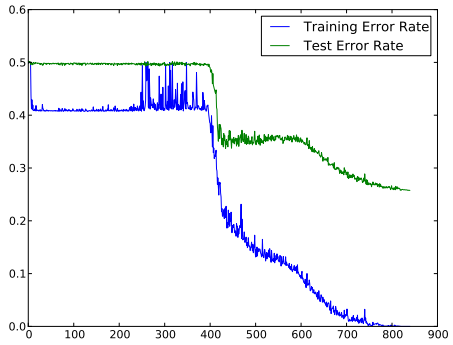
Table 2: The error percentages with different learning algorithms on pentomino dataset with different number of training examples.

Experiment 2-Disentangled representations: In this experiment, we have done trials with 16 binary inputs per patch, 10 one-hot bits for representing each object category, 4 for rotations and 2 for scaling.

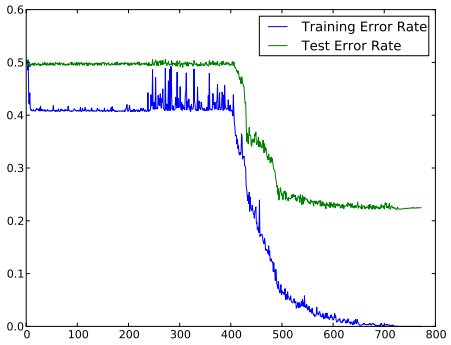
Experiment 3-Onehot representation with transformations: For each of the ten object types we have $8 = 4 \times 2$ possible transformations. Two objects in two different patches are the same if their category is the same regardless of the transformations. The one-hot representation of a patch corresponds to the cross-product between the object classes and transformations, i.e., one out of $80 = 10 \times 4 \times 2$ possibilities represented in an 80-bit one-hot vector.

Experiment 4-Onehot representation with 80 choices: This representation has the same 1 of 80 one-hot representation per patch but the target task is defined differently. Two objects in two different patches are considered the same iff they have exactly the same onehot representation (i.e., are of the same object category with the same transformation applied).

The first experiment is a sanity check. It was conducted with single hidden-layered MLP's with rectifier and tanh nonlinearity, and the task was learned perfectly (0 error on both training and test dataset) with very few training epochs.



(a) Training and Test Errors for Experiment 4



(b) Training and Test Errors for Experiment 3

Figure 4: Left (a): The training and test errors of Experiment 3 over 800 training epochs with 100k training examples.
Right (b):The training and test errors of Experiment 4 over 700 training epochs with 100k training examples.

The results of Experiment 2 are given in Table 3. We have used a Maxout non-linearity in an MLP (Goodfellow *et al.*, 2013) with two hidden layers. But unlike ordinary Maxout network mentioned in the paper, we did not use any regularization i.e: no weight decay, norm constraint on the weights, or dropout. Although learning from a disentangled representation is more difficult than learning from perfect object detectors, it is feasible with some architectures such as the Maxout network. Note

LEARNING ALGORITHM	TRAINING ERROR	TEST ERROR
SVM	0.0	35.6
RANDOM FORESTS	1.29	40.475
MAXOUT MLP	0.0	0.0

Table 3: Performance of different learning algorithms on disentangled representation (Experiment 2).

LEARNING ALGORITHM	TRAINING ERROR	TEST ERROR
SVM	11.212	32.37
RANDOM FORESTS	24.839	48.915
TANH MLP	0.0	22.475

Table 4: Performance of different learning algorithms using a dataset with onehot vector and 80 inputs as discussed in Experiment 3.

that this representation is the kind of representation that one could hope an unsupervised learning algorithm could discover, at best, as argued in Bengio *et al.* (2012a).

The best results obtained on validation set for Experiment 3 and Experiment 4 are shown respectively in Table 4 and Table 5. In these experiments we trained a tanh MLP with two hidden layers with the same hyperparameters. In experiment 3 the complexity of the problem comes from the transformations (8=4$\times$2) and the number of object types. But in experiment 4, the only source of complexity of the task come from the number of different object types. These results are in between the complete failure and complete success observed with other experiments, suggesting that the task could become solvable with better training or more training examples. Figure 4 illustrates the progress of training, on both the training and test error, for Experiments 3 and 4. Clearly, something has been learned, but the task is not nailed yet. Moreover as seen from the curves in Figure 4(a) and 4(b) the training and test error curves are almost the same for both tasks. This implies that for onehot inputs, whether you increase the number of possible transformations for each object or the number of object categories, as soon as the number of possible configurations is same, the complexity of the problem is almost the same for MLP.

4 Experimental Results and Analysis

This section provides results of experiments performed on the pentomino dataset, with different number of training examples and aimed at observing the effect of introducing intermediate knowledge. For the experimental results shown on 2 we have used 3 training set sizes (20k, 40k and 80k examples), generated with different random seeds (so they do not overlap). Figure 5 shows the error bars for an ordinary MLP with three hidden layers. For that MLP, the number of training epochs is 8 (more did not help), and there are three hidden layers with 2048 feature detectors. The learning rate we used in our experiments is 0.01. The activation function of the MLP is tanh nonlinearity and L1, L2 penalty coefficients are both 1e-6.

The P1NN learns to classify patches with respect to the intermediate concepts very quickly. According to our experiments, 5,000 training examples are enough for the P1NN to learn the intermediate targets perfectly. P2NN is then trained of top of P1NN and performs almost perfectly, both on the training set and the test set, with 80,000 examples.

Table 2 shows that, without guiding hints, none of the state-of-art learning algorithms could perform noticeably better than a random predictor on the test set. This shows the importance of intermediate hints introduced in the IKGNN. The decision trees and SVMs can overfit the training set but they could not generalize on the test set. Note that the numbers reported in the table are for hyperparameters selected based on validation set error, hence lower training errors are possible if avoiding all regularization and taking large enough models. On the training set, the MLP with two large hidden layers (several thousands) could reach nearly 0% training error, but still did not manage to achieve good test error.

LEARNING ALGORITHM	TRAINING ERROR	TEST ERROR
SVM	4.346	40.545
RANDOM FORESTS	23.456	47.345
TANH MLP	0	25.8

Table 5: Performance of different algorithms using a dataset with onehot vector and 80 binary inputs as discussed in Experiment 4.

5 Conclusion and Discussion

In this paper we have shown an example of task which seems almost impossible to solve by standard black-box machine learning algorithms, but can be almost perfectly solved when introducing an intermediate pre-trained representation guided by prior knowledge. The task has the particularity that it is defined by the composition of two non-linear sub-tasks (object detection on one hand, and a non-linear logical operation similar to XOR on the other hand).

What is interesting is that in the case of the neural network, we can compare two networks with exactly the same architecture but a different pre-training, one of which uses the known intermediate concepts to teach an intermediate representation to the network. Without using these intermediate targets the neural networks trained in our experiments have failed to learn the task. With enough capacity and training time they can *overfit* but did not not capture the essence of the task, as seen by test set performance. It could be that by training with many more examples, such high-capacity networks could eventually nail the task, and future work should investigate that. We know that a structured deep network can learn the task if initialized in the right place, and do it from very few training examples. So it looks like there is an issue that is not clearly just one of optimization nor just one of regularization. Instead, we would characterize it as one of *effective local minima with poor generalization error*. What we hypothesize is that for most initializations and architectures (in particular the fully-connected ones), although it is possible to find a *good local minimum of training error* when enough capacity is provided, it is difficult (without the proper initialization) to find a good local minimum of generalization error. On the other hand, when the network architecture is constrained enough but still allows it to represent a good solution (like the structured MLP of our experiments), it seems that the optimization problem is much more difficult and even training error remains stuck high. It could be that the combination of the network architecture and training procedure produces a training dynamics that tends to yields into these minima that are poor from the point of view of generalization error, even when they manage to nail training error by providing enough capacity. Of course, as the number of examples increases, we would expect this discrepancy to decrease, but then the optimization problem could still make the task unfeasible in practice. Note however that our preliminary experiments with increasing the training set size (8-fold) for MLPs did not reveal signs of potential improvements in test error yet, as shown in Figure 5. Future work should therefore investigate training in an online mode, i.e., using a virtually infinite training set.

These findings bring supporting evidence to the "Guided Learning Hypothesis" and "Deeper Harder Hypothesis" from Bengio (2013): higher level abstractions, which are expressed by composing simpler concepts, are more difficult to learn (with the learner often getting in an effective local minimum), but that difficulty can be overcome if another agent provides hints of the importance of learning other, intermediate-level abstractions which are relevant to the task.

Many interesting questions remain open. Would a network without any guiding hint eventually find the solution with a enough training time and/or with alternate parametrizations? Is ill-conditioning also an issue? Clearly, one can reach good solutions from an appropriate initialization, pointing in the direction of a local minima issue, but it may be that good solutions are also reachable from other initializations, albeit going through a tortuous ill-conditioned path in parameter space. Why did our attempts at learning the intermediate concepts in an unsupervised way fail? Are these results specific to the task we are testing or a limitation of the unsupervised feature learning algorithm tested? Trying with many more unsupervised variants and exploring explanatory hypotheses for the observed failures could help us answer that. Finally, and most ambitious, can we solve these kinds of problems if we allow a community of learners to collaborate and collectively discover and combine partial solutions in order to obtain solutions to more abstract tasks like the one presented here?

Indeed, we would like to discover learning algorithms that can solve such tasks without the use of prior knowledge as specific and strong as the one used in the IKGNN here. These experiments could be inspired by and inform us about potential mechanisms for collective learning through cultural evolutions in human societies.

References

Ben-Hur, A. and Weston, J. (2010). A user's guide to support vector machines. *Methods in Molecular Biology*, **609**, 223–239.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, **2**(1), 1–127. Also published as a book. Now Publishers, 2009.

Bengio, Y. (2013). Evolving culture vs local minima. In *Growing Adaptive Machines: Integrating Development and Learning in Artificial Neural Networks*, number also as ArXiv 1203.2990v1, pages T. Kowaliw, N. Bredeche & R. Doursat, eds. Springer-Verlag.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009a). Curriculum learning. In *ICML'09*.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009b). Curriculum learning. In L. Bottou and M. Littman, editors, *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*. ACM.

Bengio, Y., Courville, A., and Vincent, P. (2012a). Unsupervised feature learning and deep learning: A review and new perspectives. Technical Report arXiv:1206.5538, U. Montreal.

Bengio, Y., Courville, A., and Vincent, P. (2012b). Unsupervised feature learning and deep learning: A review and new perspectives. Technical report, arXiv:1206.5538.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*.

Breiman, L. (2001). Random forests. *Machine Learning*, **45**(1), 5–32.

Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2010). Deep big simple neural nets for handwritten digit recognition. *Neural Computation*, **22**, 1–14.

Dawkins, R. (1976). *The Selfish Gene*. Oxford University Press.

Duchi, J., Hazan, E., and Singer, Y. (2010). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, **12**, 2121–2159.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, **11**, 625–660.

Fleuret, F., Li, T., Dubout, C., Wampler, E. K., Yantis, S., and Geman, D. (2011). Comparing machines and humans on a visual categorization test. *Proceedings of the National Academy of Sciences*, **108**(43), 17621–17625.

Glorot, X., Bordes, A., and Bengio, Y. (2011a). Deep sparse rectifier neural networks. In *AISTATS'2011*.

Glorot, X., Bordes, A., and Bengio, Y. (2011b). Deep sparse rectifier neural networks. In *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*.

Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. Technical report, Université de Montréal.

Henrich, J. and McElreath, R. (2003). The evolution of cultural evolution. *Evolutionary Anthropology: Issues, News, and Reviews*, **12**(3), 123–135.

Hinton, G. E., Osindero, S., and Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, **18**, 1527–1554.

Hsu, C., Chang, C., Lin, C., *et al.* (2003). A practical guide to support vector classification.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *ICCV'09*.

Khan, F., Zhu, X., and Mutlu, B. (2011). How do humans teach: On curriculum learning and teaching dimension. In *Advances in Neural Information Processing Systems 24 (NIPS'11)*, pages 1449–1457.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*.

Krueger, K. A. and Dayan, P. (2009). Flexible shaping: how learning in small steps helps. *Cognition*, **110**, 380–394.

Kunapuli, G., Bennett, K., Maclin, R., and Shavlik, J. (2010). The adviceptron: Giving advice to the perceptron. *Proceedings of the Conference on Artificial Neural Networks In Engineering (ANNIE 2010)*.

Larochelle, H., Bengio, Y., Louradour, J., and Lamblin, P. (2009). Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, **10**, 1–40.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

Mitchell, T. (1980). *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ.

Mitchell, T. and Thrun, S. (1993). Explanation-based neural network learning for robot control. *Advances in Neural information processing systems*, pages 287–287.

Montague, R. (1970). Universal grammar. *Theoria*, **36**(3), 373–398.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *ICML'10*.

Olshen, L. and Stone, C. (1984). Classification and regression trees. *Belmont, Calif.: Wadsworth*.

O'Sullivan, J. (1996). Integrating initialization bias and search bias in neural network learning.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., *et al.* (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, **12**, 2825–2830.

Peterson, G. B. (2004). A day of great illumination: B. F. Skinner's discovery of shaping. *Journal of the Experimental Analysis of Behavior*, **82**(3), 317–328.

Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML'2011*.

Rifai, S., Bengio, Y., Dauphin, Y., and Vincent, P. (2012). A generative process for sampling contractive auto-encoders. In *ICML'2012*.

Salakhutdinov, R. and Hinton, G. (2009). Deep Boltzmann machines. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, volume 8.

Skinner, B. F. (1958). Reinforcement today. *American Psychologist*, **13**, 94–99.

Solomonoff, R. (1989). A system for incremental learning based on algorithmic probability. In *Proceedings of the Sixth Israeli Conference on Artificial Intelligence, Computer Vision and Pattern Recognition*, pages 515–527. Citeseer.

Towell, G. and Shavlik, J. (1994). Knowledge-based artificial neural networks. *Artificial intelligence*, **70**(1), 119–165.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, **11**, 3371–3408.

Weston, J., Ratle, F., and Collobert, R. (2008). Deep learning via semi-supervised embedding. In W. W. Cohen, A. McCallum, and S. T. Roweis, editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1168–1175, New York, NY, USA. ACM.

6 Appendix

6.1 Experimental Setup and Hyper-parameters

6.1.1 Decision Trees

We used the decision tree implementation in the scikit-learn (Pedregosa *et al.*, 2011) python package which is an implementation of the CART (Regression Trees) algorithm. The CART algorithm constructs the decision tree recursively and partitions the input space such that the samples belonging to the same category are grouped together (Olshen and Stone, 1984). We used The Gini index as the impurity criteria. We evaluated the hyper-parameter configurations with a grid-search. We cross-validated the maximum depth (max_depth) of the tree (for preventing the algorithm to severely overfit the training set) and minimum number of samples required to create a split (min_split). 20 different configurations of hyper-parameter values were evaluated. We obtained the best validation error with $max\_depth = 300$ and $min\_split = 8$.

6.1.2 Support Vector Machines

We used the "Support Vector Classifier (SVC)" implementation from the scikit-learn package which in turn uses the libsvm's Support Vector Machine (SVM) implementation. Kernel-based SVMs are non-parametric models that maps the data into a high dimensional space and separate different classes with hyperplane(s) such that the support vectors for each category will be separated by a large margin. We cross-validated three hyper-parameters of the model using grid-search: C, γ and the type of kernel($kernel\_type$). C is the penalty term (weight decay) for the SVM and γ is a hyper-parameter that controls the width of the Gaussian for the RBF kernel. For the polynomial kernel, γ controls the flexibility of the classifier (degree of the polynomial) as the number of parameters increases (Hsu *et al.*, 2003; Ben-Hur and Weston, 2010). We evaluated forty-two hyper-parameter configurations. That includes, two kernel types: $\{RBF,\ Polynomial\}$; three gammas: $\{1e-2,\ 1e-3,\ 1e-4\}$ for the RBF kernel, $\{1, 2, 5\}$ for the polynomial kernel, and seven C values: $\{0.1, 1, 2, 4, 8, 10, 16\}$ values. As a result of the grid search and cross-validation, we have obtained the best test error by using the RBF kernel, with $C = 2$ and $\gamma = 1$.

6.1.3 Multi Layer Perceptron

We have our own implementation of Multi Layer Perceptron based on the Theano (Bergstra *et al.*, 2010) machine learning libraries. We have selected 2 hidden layers, the rectifier activation function, and 2048 hidden units per layer. We cross-validated three hyper-parameters of the model using random-search, sampling the learning rates ϵ in log-domain, and selecting $L1$ and $L2$ regularization penalty coefficients in sets of fixed values, evaluating 64 hyperparameter values. The range of the hyperparameter values are $\epsilon \in [0.0001, 1]$, $L1 \in 0., 1e-6, 1e-5, 1e-4$ and $L2 \in 0, 1e-6, 1e-5$. As a result of have selected $L1 = 1e-6$, $L2 = 1e-5$ and $\epsilon = 0.05$.

6.1.4 Random Forests

We used scikit-learn's implementation of "Random Forests" decision tree learning. The Random Forests algorithm creates an ensemble of decision trees by randomly selecting for each tree a subset of features and apply bagging to combine the individual decision trees (Breiman, 2001). We have used grid-search and cross-validated the max_depth, min_split, and number of trees ($n\_estimators$). We have done the grid-search on the following hyperparameter values, $n\_estimators \in \{5, 10, 15, 25, 50\}$, $max\_depth \in \{100, 300, 600, 900\}$, and $min\_splits \in \{1, 4, 16\}$. We obtained the best validation error with $max\_depth = 300$, $min\_split = 4$ and $n\_estimators = 10$.

6.1.5 k-Nearest Neighbors

We used scikit-learn's implementation of k-Nearest Neighbors (k-NN). k-NN is an instance-based, lazy learning algorithm that selects the training examples closest in Euclidean distance to the input query. It assigns a class label to the test example based on the categories of the k closest neighbors. The hyper-parameters we have evaluated in the cross-validation are the number of neighbors (k) and $weights$. The $weights$ hyper-parameter can be either "uniform" or "distance". With "uniform", the value assigned to the query point is computed by the majority vote of the nearest neighbours. With "distance", each value assigned to the query point is computed by weighted majority votes where the weights are computed with the inverse distance between the query point and the neighbors. We have used $n\_neighbours \in \{1, 2, 4, 6, 8, 12\}$ and $weights \in \{"uniform", "distance"\}$ for hyper-parameter search. As a result of cross-validation and grid search, we obtained the best validation error with $k = 2$ and $weights$="uniform".

6.1.6 Convolutional Neural Nets

We used a Theano (Bergstra *et al.*, 2010) implementation of Convolutional Neural Networks (CNN) from the deep learning tutorial at `deeplearning.net`, which is based on a vanilla version of a CNN LeCun *et al.* (1998). Our CNN has two convolutional layers. Following each convolutional layer, we have a max-pooling layer. On top of the convolution-pooling-convolution-pooling layers there is an MLP with one hidden layer. In the crossvalidation we have sampled 36 learning rates in log-domain in the range $[0.0001, 1]$ and the number of filters from the range $[10, 20, 30, 40, 50, 60]$ uniformly. For the first convolutional layer we used 9×9 receptive fields in order to guarantee that

each object fits inside the receptive field. The number of features for the first layer is 30. For the second convolutional layer, we used 7×7 receptive fields with 60 features. The stride for both convolutional layers is 1. We downsample convolved images by a factor of 2 after each pooling operation. The selected learning rate for the CNN is 0.01 and we have used 8 training epochs.

6.1.7 Stacked Denoising Autoencoders

Denoising Autoencoders (DA) are a form of regularized auto-encoder (Bengio *et al.*, 2012b). The DA forces the hidden layer to discover more robust features and prevent it from simply learning the identity by reconstructing the input from a corrupted version of it (Vincent *et al.*, 2010). We used Stacked DA's, stacking two DAs, resulting in an unsupervised transformation with two hidden layers. Parameters of all layers are then fine-tuned with supervised fine-tuning using logistic regression as the classifier and SGD as the gradient-based optimization algorithm. We used 1024 hidden units and 0.2 as the corruption level with binomial corruption. We've manually tried different learning rates for the DA and the supervised fine-tuning. The selected learning rate is $\epsilon_0 = 0.01$ for DA and $\epsilon_1 = 0.1$ for supervised fine-tuning. Both L1 and L2 penalty for DA's and logistic regression are set to 1e-6.

CAE+MLP with Supervised Finetuning: Let's formulize the notation we are going to use for in general autoencoders and contractive autoencoder based on Rifai *et al.* (2012). From an input $x \in [0,1]^d$, a k-dimensional feature vector is computed as a hidden layer, e.g., $h = f(x) = s(Wx + b_h)$, where s is the element-wise logistic sigmoid. From hidden representation h, a *reconstruction* of x is obtained as $r = g(f(x)) = s(W^T f(x) + b_r)$, where $b_r \in \mathbb{R}^d$ is the reconstruction bias vector.

A *reconstruction loss* $L(x, r)$ measures how well the input is reconstructed from the hidden representation. Following Rifai *et al.* (2011), we used a cross-entropy loss, $L(x, r) = -\sum_{i=1}^{d} x_i \log(r_i) + (1 - x_i) \log(1 - r_i)$.

The training objective being minimized in a traditional auto-encoder is simply the average reconstruction error over a training set $\mathcal{D}$. In the supervised finetuning phase we use the adagrad method to automatically tune the learning rate (Duchi *et al.*, 2010).

In a nutshell, the parameters θ of the CAE are learned by minimizing:

$$\mathbb{J}_{\mathrm{CAE}}(\theta; \mathcal{D}) = \sum_{x \in \mathcal{D}} \left(L(x, g(f(x))) + \lambda \|J(x)\|^2 \right) \tag{1}$$

After training the autoencoder on each patch, we aggregated the features extracted on each patch by concatenating the activations of the autoencoder on each patch. The concatenated hidden units are added as a hidden layer to a MLP.

We used 100 hidden units for the CAE and chose contraction level λ in Equation 1 as 2. The learning rate for pretraining is 0.082 with batch size of 200 and performing 200 passes on the training dataset. The learning rate for supervised finetuning is 0.12 and L1 and L2 regularization penalty terms respectively are 1e-4 and 1e-6. We used 6400 hidden units for the top-level MLP and trained the whole architecture for 100 epochs.

Greedy Layerwise CAE+DAE Supervised Finetuning: A denoising autoencoder is trained to reconstruct a clean input from a corrupted version of it. This is done by first corrupting the input $\mathbf{x}$ and obtain $\tilde{\mathbf{x}}$ using a stochastic mapping. In order to corrupt inputs we added binomial noise on the inputs.

In our experiments we have used rectifier nonlinearity and quadratic error with denoising autoencoders with L1 sparsity and L2 penalty on the weights. Therefore our activation function is:

$$h = f(\tilde{\mathbf{x}}) = max(W\tilde{\mathbf{x}} + b_h, 0),$$

As recommended by Glorot *et al.* (2011b) we have used softplus nonlinearity for reconstruction, $softplus(x) = log(1 + e^x)$:

$$r = g(f(x)) = softplus(W^T f(x) + b_h)$$

The training objective with quadratic error loss function for the denoising autoencoder will be:

$$\mathbb{J}_{\mathrm{DAE}}(\theta; \mathcal{D}) = \sum_{\tilde{\mathbf{x}} \in \tilde{\mathcal{D}}} \parallel \tilde{\mathbf{x}} - r \parallel^2 + \lambda_1 \parallel W \parallel_1 + \lambda_2 \parallel W \parallel_2^2$$

We used L1 penalty to obtain a sparser representation with rectifier non-linearity and L2 regularization to keep the non-zero weights small.

The main difference between DAE and CAE is that, DAE yields more robust reconstruction whereas CAE obtains more robust features (Rifai *et al.*, 2011).

As seen on Figure 3 the weights U and V are shared on each patch and we concatenate the outputs of the last autoencoder on each patch to feed it as an input to a MLP with large hidden layer.

We used 400 hidden units for CAE and 100 hidden units for DAE. The learning rate used for CAE is 0.82 and DAE is 9*1e-3. Corruption level for DAE is 0.25 and contraction level for CAE is 2.0. L1 regularization penalty for DAE is 2.25*1e-4 and L2 is 9.5*1e-5. For the supervised finetuning phase the learning rate used is 4*1e-4 with L1 and L2 penalties respectively are 1e-5 and 1e-6. We used 6400 hidden units for the top level MLP. We trained the autoencoders for 150 epochs during the pretraining phase and trained the whole MLP for 50 epochs in the supervised finetuning phase.

Greedy Layerwise DAE+DAE Supervised Finetuning: In this architecture, we have trained two layers of denoising autoencoders greedily and did supervised finetuning after the pretraining ends. The motivation of using two denoising autoencoders is the fact that rectifier nonlinearities work well with the deep networks. We have used the same type of denoising autoencoder that is used for greedy layerwise CAE+DAE supervised finetuning experiment.

In this experiment we have used 400 hidden units for the first layer DAE and 100 hidden units for the second layer DAE. The other hyperparameters for DAE and supervised finetuning are same with the *CAE+DAE MLP Supervised Finetuning* experiment.

6.2 Additional Experimental Results

In the experiment results shown in Figure 5, we evaluate the impact of adding more training data for the fully-connected MLP. As mentioned before for these experiments we have used a MLP with three hidden layers where each layer has 2048 hidden units. $tanh(.)$ activation function is used with 0.05 learning rate and minibatches of size 200.

As can be seen from the figure, adding more training examples did not help either training or test error (both are near 50%, with training error slightly lower and test error slightly higher), reinforcing the hypothesis that the difficult encountered is one of optimization, not of regularization.

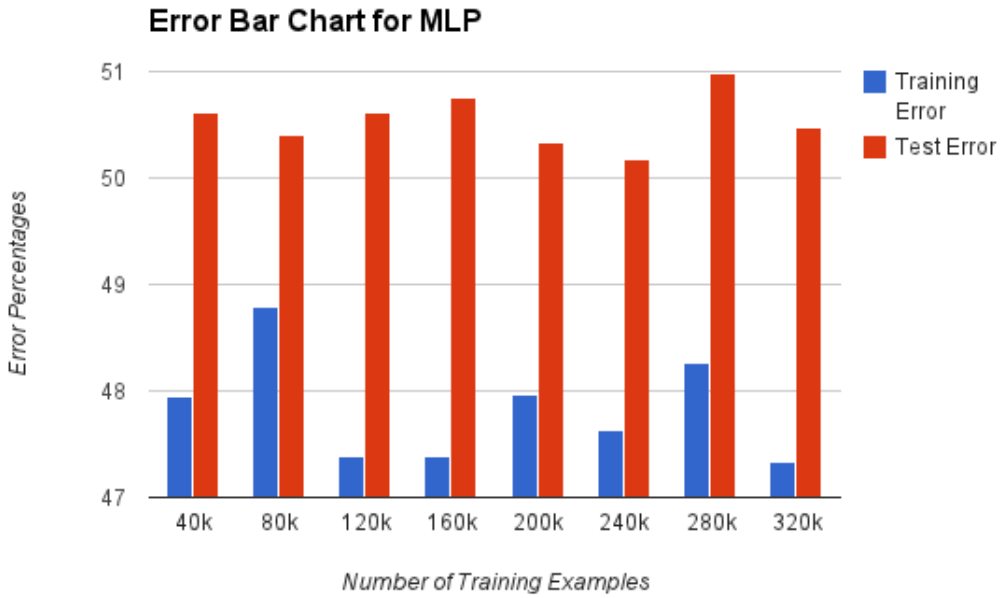


Figure 5: Training and test error bar charts for a regular MLP with 3 hidden layers. There is no significant improvement on the generalization error of the MLP as the new training examples are introduced.